RHB+: A Type-Oriented ILP System Learning from Positive Data

Yutaka Sasaki and Masahiko Haruno
NTT Communication Science Laboratories
1-1 Hikari-no-oka, Yokosuka, 239 Japan
{sasaki, haruno}@cslab.kecl.ntt.co.jp

Abstract

This paper presents the *type-oriented relational learner RHB+.* Attaching type information to hypotheses is effective in avoiding overgeneralization as well as enhancing readability and comprehensibility. In many areas, such as NLP, type information is actually available, while negative examples are not. Unfortunately, learning performance is usually poor if types are attached when only positive examples are available. *RHB+* makes use of type information to efficiently compute informativity from positive examples only and to judge a stopping condition. The new technique of *dynamic type restriction by positive examples* lets covered positive examples decide the types appropriate for the current clause. The current version of *RHB+*, written in the typed logic programming language LIFE, directly manipulates types as structured background knowledge when operations related to types are required. These features make *RHB+* efficient and effective in attaching types selected from thousands of possible types. This leads to advantages over several previous learners, such as FOIL and PROGOL. Experimental results demonstrate *RHB+* 's fine performance for both artificial and real data.

1 Introduction

Inductive Logic Programming (ILP) [Muggleton, 1991] is a promising approach to *knowledge-level learning* from real-world examples such as the wide variety of facts contained in on-line newspapers or large-scale databases. Major extensions related to *types* (or *sorts)* are, however, required for ILP systems to handle real-world relations. First, types should be always attached to hypotheses in order to make comprehensible rules with appropriate generality. Second, background knowledge, even if it includes thousands of hierarchically structured types, must not deteriorate system performance. These extensions have to be achieved within the restriction that only positive examples are available because negatives are not usually found in real-world data. Actually, *type hierarchy* has played a key role in several natural language processing systems, such as [Ikehara et al., 1993], [Hastings and Lytinen, 1994], Knowledge representation languages [Ai't-Kaci and Nasr, 1986], [Ai't-Kaci et al., 1994], [Borgidaand Patel-Schneider, 1994], [Sowa, 1991] also incorporate types and they achieve efficient *inheritance* according to the type hierarchy used.

The following example illustrates that type information is essential to learning in the situation in which a human could learn rules of appropriate generality from a small number of examples by using type hierarchy.

Example 1 (*Sample data of "speak"*)

Suppose that a person knows the following relations :

- *Positive example :*
  { *speak*(*Jack,English*), *speak*(*Jun,Japanese*) }
- *Background knowledge :*
  { *grew_in*(*Jack, UK*), *grew_in*(*Jun,Japan*), *official_lang*(*UK,English*), *official_lang*(*Japan,Japanese*) }.

Existing ILP systems may produce the clause :

$$speak(X,Y) :- grew\_in(X,Z),\ official\_lang(Z,Y).$$

This clause is, however, over-general because it says that everything that grew up in a country, even a dog or cat, speaks that language. Types effectively solve this problem. In this case, we expect the following relations with types to be produced:

$$speak(X{:}human, Y{:}language) :- grew\_in(X, Z{:}country),\ official\_lang(Z,Y),$$

where *Var:T* denotes that every variable *Var* appearing in a clause is the type *T*. In other words, this clause represents "A human speaks a particular language if s/he grew up in a country whose official language is that language".

To learn this relation, we have to give the following additional background knowledge to the learner:

{ *Jack* < *male*, *Jun* < *female*, *male* < *human*, *female* < *human*, *English* < *language*, *UK* < *country*, *Japanese <language, Japan <country }*

Note that $A < B$ denotes A is a sub-type of B.

Adding *is_a* relations to background knowledge seems to lead us to our goal but conventional ILP systems virtually fail to produce hypotheses with appropriate types even though background knowledge incorporates types and *is_a* relations. This is mainly because it is not realistic to handle a large number of fragments of a large-scale type hierarchy as ordinary background knowledge. The types have a tree or lattice structure and giving right types means to follow the links of the structure based on the generalization and specialization operations.

In this paper, we present the design and implementation of the type-oriented ILP system RHB^+ to solve the problems. In Section 2, the algorithms and implementation methods of *RHB+,* especially the novel mechanisms for learning relations with types, are described. Experimental results are shown in Section 3. Section 4 describes related work and Section 5 concludes this paper.

2 *RHB+*

In this section, we describe the novel relational learner RHB^+ which generates typed Prolog programs from just positive examples on the basis of background knowledge which might include a large-scale type hierarchy.

2.1 Hypothesis Language

The hypothesis language of *RHB+* is the Horn clause [Lloyd, 1987] based on the restricted form of ψ-terms used in LOGIN [Ai't-Kaci and Nasr, 1986] or LIFE [Ai't-Kaci et al., 1994][1]. For convenience, we call this restricted form r-terms. Informally, r-terms are Prolog terms whose variables are replaced with variable *Var* of type T, which is denoted as *Var:T.* Moreover, a function is allowed to have labels or keywords for readability. The definition of r-terms is as follows.

Definition 1 (*τ-terms*)

1. *An individual constant is a term.*
2. *If τ is a type and X is a variable, $X:\tau$ is a term.*
3. *If $t_1, ..., t_n$ are terms, $l_1, ..., l_n$ are labels, and f is a function symbol of arity n, then $f(l_1 \Rightarrow t_1, ..., l_n \Rightarrow t_n)$ is a term.*

Labels are symbols used as keywords to indicate the content of arguments in terms. For example, *speak(name ⇒Jack, lang ⇒French)* is a literal based on τ-terms whose labels are "name" and "lang". While τ-terms have labels, they are compatible to usual Prolog terms. A usual logic term can be regarded as a τ-term whose labels are argument position numbers in the terms. For example,

got(Jack, good_grade)

[1]wild_life, an implementation of LIFE, is available at http://www.isg.sfu.ca/life

is regarded as

got(1 ⇒Jack, 2 ⇒ good_grade)

and sometimes the former is used for the concise representation of a τ-term when the labels are numbers.

As an abbreviation, $X:\tau$ is denoted as τ if X is bound to no other variable or constant. $X:\tau$ is denoted as X if τ is the universal type (*i.e.*, the root of a type hierarchy) or if $X:\tau$ appeared before in a clause.

For example,

happy(X:human) :– got(X:human, Y:good_grade)

can be abbreviated as

happy(X:human) :– got(X, good_grade),

where *human* and *good_grade* are types.

2.2 Lgg with types

In the definition of *the least general generalization (lgg)* [Plotkin, 1969], the definition of the term *lgg* should be extended to τ-term *lgg.* Other definitions of *lgg* are equivalent to the originals. Note that $lub(s,t)$ is the function which returns the *least upper bound* of s and t according to a type hierarchy if t and s are types, otherwise it returns a new variable X.

Definition 2 (*τ-term lgg*)

1. $lgg(t,t) = t$.
2. $lgg(f(l_1 \Rightarrow s_1, .., l_n \Rightarrow s_n), f(l_1 \Rightarrow t_1, .., l_n \Rightarrow t_n) = f(l_1 \Rightarrow lgg(s_1,t_1), ..., l_n \Rightarrow lgg(s_n,t_n))$.
3. *$lgg(s,t) = u$ for term s and t of different symbols or with different function names, where the tuple (s,t,u) is in the history H.*
4. *$lgg(s,t) = u$ for term s and t of different symbols or with different function names, where $u = lub(s,t)$ and (s,t,u) is not in H. Then add (s,t,u) into H.*

RHB^+ employs this *lgg* to make candidates for the heads of hypothesis clauses from pairs of positive examples.

For example,
lgg(person(id ⇒ Jack, father ⇒ Ken),
person(id ⇒ Jun, father ⇒ Ken))
= person(id ⇒ human, father ⇒ Ken).

2.3 *RHB+* Algorithms

Given positive examples are non-negative literals based on r-terms. The hypothesis language is Horn clauses based on r-terms. The background knowledge are non-negative literals.[2] *RHB+* employs a combination of bottom-up and top-down approaches, following the result described in [Zelle *et* a/., 1994]. That is, first make the head in a bottom-up manner then construct the body in a top-down manner.

The outer loop of *RHB+* finds covers of the given positive examples *P* in a greedy manner. It constructs

[2]Horn clauses that produce finite numbers of atoms can be included.

clauses one by one by calling *inner Joop(P,Po,BK)* which returns a hypothesis clause, where P_o is original positive examples and *BK* is background knowledge. Covered examples are removed from *P* in each cycle; P_0 remains unchanged.

Algorithm 1 *inner Joop(P,P_0,BK)*

```
1. Given positive examples P,P_0, background knowledge BK.
2. Decide types of variables in a head by computing typed lggs of N pairs of elements in P, and select the most general head as Head.
3. If StopCond(Po, BK, Head) is satisfied, return Head.
4- Let Body be empty.
5. Create a set of all possible literals L using variables in Head and Body.
6. Let BEAM be top K literals $l_k$ of L with respect to positive weighted informativity, PWI(P,BK,(Head:-Body,$l_k$)).
7. Do later steps, assuming that $l_k$ is added to Body for each literal $l_k$ in BEAM.
8. Dynamically restrict types in Body by calling restrict (P,BK,(Head :- Body)).
9. If StopCond(Po,BK,(Head :- Body)) is satisfied, return (Head :- Body).
10. Goto 5.
```

We will now see how types are utilized in each component of the *RHB*+ algorithm in the following sections.

2.4 Dynamic type restriction by positive examples

The special feature of *RHB*+ is the *dynamic type restriction by positive examples* during clause construction. *restrict(P,BK,(Head :-Body))* in Algorithm 1 does this part, where *P* represents positives, *BK* is background knowledge, and *(Head:—Body)* is a hypothetical clause. The restriction uses positive examples currently covered in order to determine appropriate types. Informally, for each variable X_i appearing in the clause, *RHB*+ computes the *lub* of all types bound to X_i when covered positive examples are unified with the current head in turn. Formally, the dynamic type restriction by positive examples is defined as follows.

Algorithm 2 *(Dynamic type restriction by positive examples)*

```
1. Given a hypothesis clause Hypo = (Head .Body) and positives P.
2. Collect all the types in Hypo and put them into a list TypeSet.
3. Let P be examples in P covered by Hypo.
4- For all elements $p_i$ of P , unify Head and $p_{if}$ then prove Body- Make a list TypeSeti of bound types in the proved Head and Body so that the position of each type in TypeSeti correctly corresponds to the position of the original type in TypeSet.
5. For all $X_k$ in TypeSet, compute lub $r_k$ of all bound types of $X_k$ in TypeSet, •
6. For all k, bind $T_k$ to $X_k$ •
```

Type restriction binds a type to each newly introduced variable in the body. Without a type restriction, newly introduced variables would always have no types, and *RHB*+ might produce over-general clauses. Note that the result of the type restriction operation by unification dynamically affects the types of all variables related to the unified variable. This operation is directly implemented by using the type unification mechanism of LIFE.

It would rather not aggressively add type to narrow the current cover but it is interesting that the dynamic type restriction significantly contributes to narrowing the current cover and helping the learner to find good hypotheses.

Example 2

When we have positive examples and background knowledge as given in Example 1 and additional data about cat Socks.

- *Additional positive example :*
  { speak(Socks, cat-lang) }
- *Additional background knowledge :*
  {grew_in(Socks, Japan), Socks < cat].

Suppose that *speak(agent, anything)* is the current head. Adding *official_lang/2* , one of the candidates for additional literal, restricts the types in the head as follows. The current clause before the type restriction is:

speak(agent, Y:anything) :— official_lang{X,Y).

The second argument of *official_lang* matches an official language. This cause that covered positives are only examples relating to humans because the positive related to *Socks* is no longer covered. Therefore, the type *agent* is restricted to humans and we obtain:

speak(human, Y-.language) :— officialJang(X,Y).

After that, the data unrelated to humans will not affect the clause construction. This illustrates how the typing contributes to restricting types in forming typed clauses.

2.5 Use of types in computing Informativity

Type information is also used to compute informativity $PWI(P,BK,(Head:-Body))$.

Let $T = \{Head:-Body\} \cup BK$.

$$PWI(T) = -|\hat{P}| \times \log_2 \frac{|\hat{P}|+1}{|Q(T)|+2},$$

where $|\hat{P}|$ denotes the number of positive examples covered by T. In effect, this is weighted informativity employing the Laplace estimate [Cestnik, 1990]. $Q(T)$ is the empirical content and defined as follows.

Definition 3 (*Empirical content*) *$Q(T)$ is a set of atoms with the predicate name of $Head$ in the least Herbrand model $M(T)$ of theroy T. $Q(T)$ is called the empirical content. This notation is borrowed from the model complexity measure [Conklin and Witten, 1994].*

Type information is very useful for computing $Q(T)$. Let Hs be a set of instances of $Head$ generated by proving $Body$ using backtracking. We introduce the notation $|\tau|$ which expresses the number of constants that type τ represents. When τ is a constant, $|\tau|$ is defined as 1. Intuitively, $|\tau|$ is the number of leaves under τ in the type hierarchy.

$$|Q(T)| = \sum_{h \in Hs} \prod_{\tau \in Types(h)} |\tau|,$$

where $Types(h)$ returns the set of types in h.

Example 3

Positive examples and background knowledge are as given in Example 1. T is the background knowledge and the following clause:

$speak(human, Y: language) :- official\_lang(X,Y).$

In this case, the set of instances Hs is $\{speak(human, Japanese), speak(human, English)\}$. Caution is needed in that $human$ is the type representing two constants: $Jack$ and Jun.

$$\begin{aligned} |Q(T)| &= |human| \times |Japanese| + |human| \times |English| \\ &= 2 \times 1 + 2 \times 1 = 4. \end{aligned}$$

Type information drastically reduces the time to calculate informativity because it cuts the effort of generating huge numbers of combinations of constants for computing the empirical content. In this case, $PWI(T)$ is calculated as follows:

$$PWI(T) = -2 \times \log_2 \frac{2+1}{4+2} = 2.$$

2.6 Use of types in the stopping condition

This section describes how the stopping condition $StopCond(P_0, BK, (Head :- Body))$ works. When given only positive examples, usual informativity can not be applied. When T is the current hypothesis and background knowledge, we use the *Model Covering Ratio (MCR)*:

$$MCR(T) = \frac{|\hat{P_0}|}{|Q(T)|}.$$

The stopping condition is "$MCR(T) \leq \alpha$" for a predefined constant α $(0 \leq \alpha \leq 1)$. Note that $|\hat{P_0}|$ here denotes the original positive examples covered by T. This is because |Q(T)| may include a lot of examples removed from *P* in earlier steps . Type information plays a key role in computing the stopping condition because it permits the efficient calculation of |Q(T)| as described in the previous section.

3 Experiments and Results

In order to confirm that *RHB+* can efficiently handle a type hierarchy, two kinds of experiments were conducted with one part of 3000 *is_a* relations. We selected more appropriate representation of *is_a* relations for each learner: PROGOL incorporated *is_a* literals, which represent direct links in a type hierarchy, in background knowledge and FOIL used type literals.

The first experiment determined the effect of type hierarchy size. We tested FOIL, PROGOL[3] and *RHB+* on artificial data while changing hierarchy size.

The second experiment measured the performance of those three learners with real data extracted from newspaper articles. We used a SparcStation 20 with 96 Mbyes of memory for the experiments.

3.1 Type Hierarchy

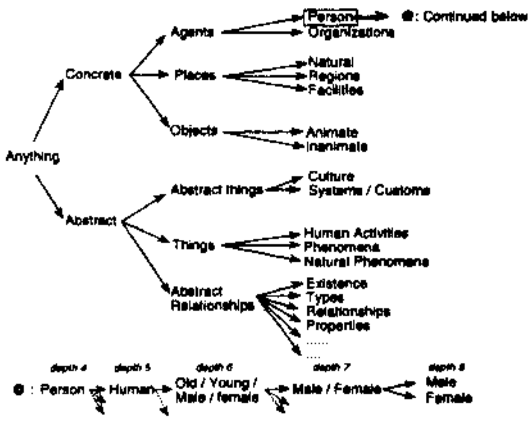

Figure 1: The upper levels of the type hierarchy

Figure 1 shows the structure of our type hierarchy [Ikehara et al., 1993]. The hierarchy is a sort of concept thesaurus represented as a tree structure in which each node is called a category *(i.e.,* a type). An edge in this structure represents an *is_a* relation among the categories. For example, "Agents" and "Person" (see Figure 1) are both categories. The edge between these two categories indicates that any instance of "Person" is also an instance of "Agents". The current version of type hierarchy is 12 levels deep and contains about 3000 category nodes. Such level of detail was found necessary to perform semantical analysis that enabled real world text understanding [Ikehara et al., 1993].

[3]PROGOL4.2 with set(posonly) option.

3.2 Learning time and type hierarchy size

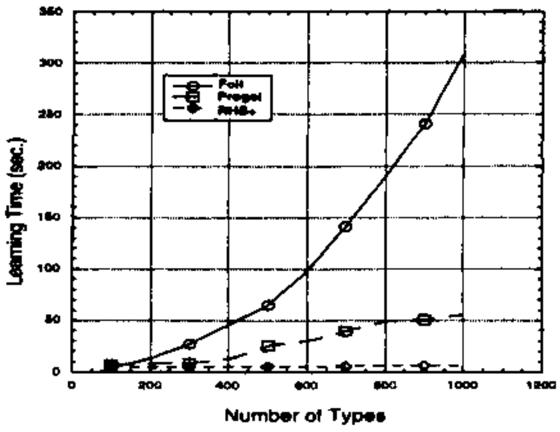

Figure 2: Learning time vs type hierarchy size

In order to estimate the effect of type hierarchy size on learning speed, we randomly generated positive examples that satisfied the following *answer clause:*

speak(A : person, B : language) :-

grew_in(A,C : country), official Jang (C, *B).*

Figure 2 shows that FOIL exponentially slows as the type hierarchy size increases. On the other hand, the learning speed of *RHB+* and PROGOL were not affected so much by the number of *is_a* relations. MCRs of the results from the artificial data should not be taken seriously.

3.3 Experiment by Real Data

We extracted articles relating to accidents from a one-year newspaper corpus written in Japanese.[4] Forty two articles are related to accidents that resulted in some numbers of death and injury. We parsed the sentences with the commercial-quality parser of our machine translation system [Ikehara et al., 1993], and performed semantic analysis to extract the case frames in the sentences. Twenty five articles were well parsed and semantically analyzed. Case frames were converted into literals and we created two kinds of positive examples from those literals. One was death.toll which represents the number of deaths in each article and the other was injury which represents the number of injuries in an article. In prepossessing, literals unrelated to positive examples were removed. The result was that three literals for each article were selected as background knowledge. The data set consisted of 25 positive examples. The background knowledge contained 78 literals with 26 kinds of predicates and 124 types and constants.

Tables 1 and 2 show the learning results of FOIL, PROGOL and *RHB+*. *\Hypo* shows the number of clauses in

[4]We used the Mainichi Newspapers articles of 1992 under appropriate permission.

Table 1: Learning results of "death toll"

Learner	Time (sec)	$\|Hypo\|$	$\|\hat{P}\| / \|Q(T)\|$
FOIL	587.4	3	25/25
PROGOL	59.1	3	25/26
RHB^+	172.7	4	25/25

Table 2: Learning results of "injury"

Learner	Time (sec)	$\|Hypo\|$	$\|\hat{P}\| / \|Q(T)\|$
FOIL	1684.5	9	24/24
PROGOL	346.9	9	25/37
RHB^+	508.5	8	25/25

hypotheses *Hypo. T* is the union of the hypotheses and background knowledge. |P|/|Q(T)| shows the MCR, the ratio between covered positives and the empirical content of T, that is, the size of the set of all instances of the head provable from T Handling type information seriously degraded the learning speed of FOIL. PROGOL was relatively fast; however it's results were over-general in both experiments. RHB^+ recorded both good performance and appropriate generality of the outputs.

Because of space limits, only the learning results from data "death toll" are shown here as follows.

• **FOIL result**

```
death_toll(A,B) :- dead(A,B,C,D).
death_toll(A,B) :- confirmed(A,C,B), anything(B).
death_toll(A,B) :- null(B), is_article_1(A).
```

• **PROGOL result**

```
death_toll(a1,'').
death_toll(A,B) :- dead(A,B,C,D).
death_toll(A,B) :- confirmed(A,C,B).
```

•RHB^+ **result**

```
death_toll(article=>_A:any, dead=>_B:number) :-
     dead(    article=>_A,
                agent=>_B,
            sub_agent=>any,
                cause=>any).
death_toll(article =>a1,dead=>'').
death_toll(article =>_A:a4, dead=>_B:four) :-
    confirmed(article=>_A, by=>accident, obj=>_B).
death_toll(article=>a23, dead=>'driver').
```

4 Related Work

Some previous learners, such as FOIL [Quinlan, 1990], GOLEM [Muggleton, 1990] and PROGOL [Muggleton, 1995] use type or sort declarations for curtailing the search space. Their learning results, however, do not have type information linked to those declarations. Simply including types and *is_a* relations in background knowledge is not a solution to obtaining typed clauses. The reason is that the possibility of a long *is_a* or type chain creates excessive overhead; the learner must search for all is_a literals or type literals. For example, when the type hierarchy is twelve levels de^p, a chain of up to twelve *is.a* literals should be checked. When *is_a* represents direct links in a type hierarchy, one possible chain might be:

is _a(X, Y),is_a{Y, Z),.., is_a{W, V),is_agent{V),....

When *is_a* includes indirect links in a type hierarchy, atoms to be checked are:

is_a(X, male), is_a(X, human),,.., is_a{X, agent),....

When type literals represent types in a type hierarchy, atoms to be checked are:

male(X),human(X), ...,agent(X),....

In those cases, top-down learners spend too much time trying to construct those chains while bottom-up learners try to remove the some of *is_a* or type atoms and find good hypotheses.

Special treatment to types was presented in [Yamazaki *et al.,* 1995]. It requires both positive and negative examples to efficiently decide *one_sa* atoms.

According to an input-output declaration, FOIDL [Mooney and Califf, 1995] generates implicit negatives by *output queries* for input arguments of positive examples in a normal ILP setting. *RHB+* utilizes type information to compute the number of covered examples including implicit negatives.

5 Conclusions and remarks

RHB+, which learns typed Prolog programs, was presented. Its performance is not affected by the number of types or type hierarchies size for the sake of direct manipulation of types and utilization of type information in computing informativity heuristics and stopping conditions. It also achieved appropriate generalization levels of hypotheses. At this point, a full LIFE compiler is not available but the current interpretive version of *RHB+* showed good performance. The execution speed will markedly improved when a LIFE compiler becomes available.

Acknowledgments

We thank the anonymous referees for their constructive comments. It helped us improve performance of FOIL and PROGOL in the experiments.

References

[Ai't-Kaci and Nasr, 1986] H. Ai't-Kaci and R. Nasr, LOGIN: A logic programming language with built-in inheritance, *J. Logic Programming,* 3, pp. 185-215, 1986.

[Ai't-Kaci et al., 1994] H. Ai't-Kaci, B. Dumant, R. Meyer, A. Podelski, and P. Van Roy, *The Wild Life Handbook,* 1994.

[Borgida and Patel-Schneider, 1994] A. Borgida and P. F. Patel-Schneider, A semantics ans complete algorithm for subsumption in the CLASSIC description logic, *Journal of Artificial Intelligence Research,* 1, pp.277-308, 1994.

[Cestnik, 1990] B. Cestnik, Estimating probabilities: A crucial task in machine learning, *ECAI-90,* pp. 147-149, 1990.

[Conklin and Witten, 1994] D. Conklin and I. H. Witten, Complexity-Base Induction, *Machine Learning,* vol.16, pp.203-225, 1994.

[Hastings and Lytinen, 1994] P. M. Hastings and S. L. Lytinen, The Ups and Downs of Lexical Acquisition, *AAAI-94,* pp.754-759, 1994.

[Ikehara et al., 1993] S. Ikehara, M. Miyazaki, and A. Yokoo, Classification of language knowledge for meaning analysis in machine translations, *Transactions of Information Processing Society of Japan,* vol. 34, pp.1692-1704, 1993 (in Japanese).

[Lloyd, 1987] J. Lloyd, *Foundations of Logic Programming,* Springer, 1987.

[Mooney and Califf, 1995] R. J. Mooney and M. E. Califf, Induction of First-Order Decision Lists: Results on Learning the Past Tense of English Verbs, *JAIR,* vol. 3, pp.1-24, 1995.

[Muggleton, 1990] S. Muggleton, Efficient induction of logic programs, *First Conference on Algorithmic Learning Theory,* Tokyo, 1990.

[Muggleton, 1991] S. Muggleton, Inductive logic programming, New Generation Computing, 8(4), pp.295-318, 1991.

[Muggleton, 1995] S. Muggleton, Inverse Entailment and Progol, *New Generation Computing Journal,* Vol. 13, pp.245-286, 1995.

[Plotkin, 1969] G. Plotkin, A note on inductive generalization, in B. Jeltzer *et al.* eds., Machine Intelligence 5, pp.153-163, Edinburgh University Press, 1969.

[Quinlan, 1990] J. R. Quinlan, Learning logical definitions from relations, *Machine Learning,* 5, 3, pp.239-266, 1990.

[Yamazaki *et al,* 1995] T. Yamazaki, M. Pazzani, and C. Merz, Learning Hierarchies from Ambiguous Natual Language Data, *ML-95,* pp.575-583, 1995.

[Sowa, 1991] J. F. Sowa *ed.,* Principle of semantic networks, Morgan Kaufmann, 1991.

[Zelle *et al,* 1994] J. M. Zelle and R. J. Mooney, J. B. Konvisser, Combining top-down and bottom-up methods in inductive logic programming, *ML-94,* pp.343-351, 1994.